A HPC Sparse Solver Interface for Scalable Multilevel Methods

Fang Liu[1], **Masha Sosonkina**[2], **Randall Bramley**[1]
1**. Computer Science Department, Indiana University**
2**. Ames Laboratory, Iowa State University**

Keywords: component architecture and interfaces; sparse matrix computations

Abstract

Sparse linear system solvers account for many of the CPU cycles in scientific computing simulations. Several HPC solver packages have been created and distributed to improve efficiency. As part of the Common Component Architecture (CCA) effort, a LInear Solver Interface (LISI) is being developed among them. LISI does not supplant HPC solvers, and the goal is to make it easier for users to switch solvers per application (run-time) demands. Some highly efficient scalable system methods, such as multigrid, multipole, or hierarchical $O(N)$ solvers, typically need more information than base LISI interface provides. The recursive features of those solvers require special care when defining software interfaces. Through subclassing LISI, a medium-level interface is designed for an application to have finer controls over an Algebraic Multigrid solver. Additionally, an interface is proposed to handle the geometric information for a Geometric Multigrid solver. For more detailed user control, a low-level interface is designed in an operator based manner. This paper presents design issues in the proposed CCA interfaces and a reference implementation for several multigrid packages. Test results show that the interface overhead is negligible while the usage complexity of the packages is reduced greatly.

1 Introduction

Sparse linear systems $Ax = b$, with $A \in R^{N\times N}$ have an important role in Partial Differential Equations (PDEs) based scientific simulation and are often the most computationally intensive part. Massively parallel computers require scalable solvers to fully utilize the available computer power and to provide higher resolution simulations. Solvers must be robust and have near-optimal computational complexity, with enough concurrency to be effectively parallelized. Multigrid (and more generally, multilevel) techniques have been proven scalable and optimal, because they can solve some linear system with N unknowns with $O(N)$ or $O(N\log(N))$ operations [8, 12], and have overhead costs that are $O(\log(p))$, where p is the number of processes used.

A common feature of many variations of multigrid is that they use a hierarchy of spatial discretizations (grids or meshes). Grids with fewer nodes are called *coarse* and ones with more nodes are called *fine*. Multigrid Method (MGM) solvers generally have more than two levels in the hierarchy. The important operations are:

- restriction, defining a related but smaller linear system and mapping a vector onto the corresponding reduced dimensionality space,
- prolongation, interpolating or extrapolating values from a smaller to a larger linear system, and
- relaxation (also called *smoothing*), executing a few steps of some iterative method on each level.

Restriction and *prolongation* transfer information between coarse and fine grids. Multigrid methods usually take a few steps of an iterative method on the fine grid, then transfer that information to the coarser grid by *restriction*; after approximately solving the coarse grid system of equations, the information is transferred back through *prolongation* and another few steps of relaxation are performed on the finer grid. The different iterative methods can be used on different levels of the mesh hierarchy. This is a typical V-cycle algorithm, which may itself be repeated iteratively if necessary [8].

Multigrid methods are either Geometric Multigrid (GMG) or Algebraic Multigrid (AMG). GMG is typically used for structured or semistructured meshes, where the geometry of the problem defines the various multigrid levels. AMG requires no explicit knowledge of the problem geometry [1] to determine coarser grids and is useful for unstructured meshes where it is impractical to construct coarser meshes geometrically. In AMG, intergrid transfer operators and coarse grid equations are based solely on the nonzero structure of A. Software interface design for AMG is more straightforward than for GMG.

Analysis of existing multigrid packages shows that a high level interface support is useful. The interface should help rapid prototyping of multigrid solvers and make application usage easier. Choosing a "best" iterative solver for an application's nonsymmetric systems of equations is difficult, and solver libraries have different strengths and weaknesses. This makes rapid plug-and-play replacing of a solver a key goal. If the number of lines of code is used as a metric to evaluate the software complexity of a package, High Performance Preconditioners (HYPRE)'s [1] Semicoarsening Multigrid (SMG) solver for a Poisson equation on structured meshes requires around 217 lines of code. Solving a 3D Poisson equation using Portable, Extensible Toolkit for Scientific Computation (PETSc)'s Data Management on Multigrid (DMMG) [5] interface involves 101 lines, and the AMG "smoothed aggregation" [18] solver from Trilinos ML[7] requires about 183 lines of code. A major advantage of high level interfaces is to ease switching between multigrid implementations, and between non-multigrid and multigrid solvers.

The CCA [9] is a software component model that specifically addresses high performance scientific applications. The interface is written in Scientific Interface Definition Language (SIDL) with a language and platform independent way, and the earlier work on proposed LISI interface [16, 17] has successfully demonstrated a generic interface *lisi.SparseSolver*, and its prototype implementation has been applied to the fusion application codes Non-Ideal Magnetohydrodynamics with Rotation - Open Discussion (NIMROD)

[4] and M3D [2]. In this paper, a subclass of LISI is designed to support more efficient multigrid solvers.

2 Interface Requirements

To identify a suitable and smallest common set of interface among widely available multigrid solver packages [1, 13, 5], some requirements are investigated first based on them.

2.1 Analysis of Multigrid Solvers

Both AMG and GMG methods have two basic computational phases, *initialization* and *iteration*. A hierarchy of interpolation matrices are generated in the initialization phase and the iterative phase performs intergrid data transfers and smoothing. In Section 1, both AMG and GMG follow the the basic multigrid solver principle differing only in the user provided information. AMG requires only the linear system to be used, while GMG needs both the linear system and geometry information such as domain size and shape, discretized mesh and stencil, etc. The construction (or initialization) phase builds the hierarchy of grids based on the physics problem, and depending on the coarsening or refinement algorithm used [10] the operators may have vastly different presentations. The solving or iterative phase has two layers. One is the outer iteration which specify a V or W cycle that can be controlled by the MGM solver either in a manner totally hidden from the application or the application may call directly each V or W cycle and control the convergence test. The second layer is the recursive solve within one V or W. This is done by a MGM solver internally but an interface can be provided to the application. Trilinos ML provides both methods: *ML_Solve_MGV* only performs one multigrid V cycle iteration and *ML_Iterate* iterates until convergence to solve the linear system. HYPRE's BOOMERAMG only allows control of the outer loops through *HYPRE_BoomerAMGSolve* method. When an interface is designed, the responsibility for these two phases is split between the application and solver side.

2.2 Interface Boundary

Since the interface sits between the application and solver packages, the division of work needs to be examed. The two solving phases for MGM discussed in Section 2.1 provide three use cases (Figures 1, 2, 3). Some applications are built

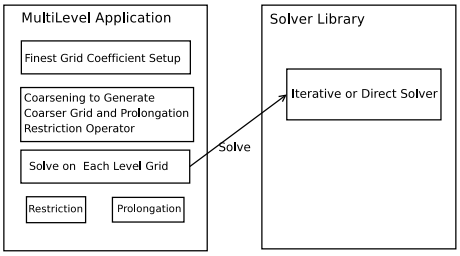

Figure 1. Application Builds the Multigrid Hierarchy and Solver

ground up for one particular purpose, and the application code has the underlying geometric, discretization operator, and physics information. It would be easy to let the application to construct the hierarchy of grids and other intergrid transformation operators and build the multigrid solver itself. In this case the only functionality needed from solver packages is to smooth the linear system on each grid level and to solve the coarsest grid. Then the application can use any solver through a LISI [16] interface (Figure 1).

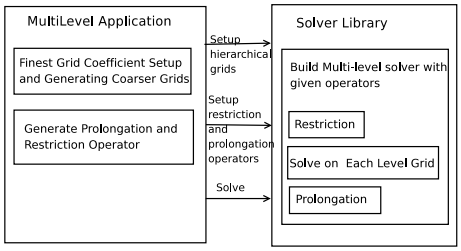

Figure 2. Application Builds the Multigrid Hierarchy

The second case has the application building the hierarchy of grids and related operators and passing these operators to a multigrid solver building framework. Some approaches [5, 3] for designing an operator based interface shows this assumption, while the MGM solver framework is used to fully utilize the internal solver functions (Figure 2). This can also happen inside the MGM solver to allow fast prototyping of a MGM algorithm. Our design effort will treat this interface as a low-level interface in which multigrid application has more controls on the solver.

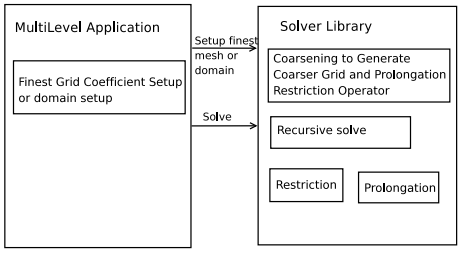

Figure 3. Application Builds the Finest Grid or Initial Domain Information

In the third case the application only specifies the domain information to allow the GMG solver to construct the coarsest or finest grid linear system followed by a hierarchy of grids, or passes the finest grid linear system to an AMG solver to construct the coarser linear systems internally (Figure 3). This assumes GMG as a black-box solver where the application requires the least effort to use the solvers. Terascale Optimal PDE Solvers — a SciDAC ISIC (TOPS) CCA interface [6] provides a high level access to geometric and algebraic multigrid, but it is a bottom up design and implementation mainly based on PETSc's DMMG interface. Our design will provide an interface specially for GMG and add more parameter control for AMG, such that it becomes the medium-level interface in LISI.

Different interfaces can be designed for all three cases. Each of them assumes a different level of application involvement. For the first case, it is the easiest to adapt LISI when the outside solver is needed. There are two approaches for the

second case demonstrating the successful operator based interface design. Based on our previous research [17], the subclassing from LISI by adding more parameter controls can be designed for the third case, while operator based interface becomes the next level hierarchy.

2.3 Matrix or Matrix-free MGM

In Section 2.2, the boundary of interface between application and solver shows that both matrix-based and matrix-free interface invocations could happen in the second case while a matrix-based version needs the explicit setting of the multigrid operators (grid hierarchy, prolongation and restriction operators) in a sparse matrix format and passed through the interface. The matrix-free version provides a call back function such as applying the given operator to a vector (e.g. *matrix-vector product*, *prolongation*, *restriction*) for the MGM solver to use. PCMG [5] allows a matrix-free operator to use their multigrid solver through *shell matrix*. But the third case from Section 2.2 may not be able to process the coarsening internally in a matrix-free fashion for both AMG and GMG solvers, so the medium-level interface need not support matrix-free MGM, and explicitly passing the matrix data is required.

2.4 Geometric Information for GMG

HYPRE's [1] structured and semi-structured grid interface provides a natural way to use GMG solver, only requiring the user to input the grid and stencil information from the physics problems. In HYPRE the domain is partitioned among the given processors, and the mesh points are identified with a global integer index. The domain owned by current processor is represented as a box with the start mesh point and end mesh point in a coordinate-wise index. The application must prepare the values to insert based on coefficient functions of PDEs. The linear system and right-hand side values are set through the conceptual interface along with geometric information so that an internal data structure is able to do coarsening with the given geometric information. HYPRE also provides an interface to set boundary values for the problem, and it is also associated with geometric information in the form of a box domain.

PETSc's DMMG [5] interface also provides an interface for an application to specify the domain through its DA object. The domain is specified by the number of mesh points in each domain direction, and only a single grid domain is supported. The stencil type *DA_STENCIL_STAR* and *DA_STENCIL_BOX* can be chosen, where the former indicates only (i, j, k), $(i \pm s, j, k)$, $(i, j \pm s, k)$, $(i, j, k \pm s)$ are in the stencil while the later allows $(i \pm s, j \pm r, k \pm t)$ to be in the stencil, and the box stencil is supported in PETSc now.

This geometric information is based only on a structured grid with finite difference discretizations. The question remains whether an interface need to be provided for more complex geometries. The number of possible discretization schemes is large, so the proposed interface focuses on currently available GMG solver packages with structured grids and finite difference schemes.

2.5 Separation of AMG and GMG

HYPRE's BoomerAMG solver has methods to set up the finest linear system in *HYPRE_ParCSRMatrix* format; the maximum number of levels of multigrid; the maximum iteration number for total V or W cycles; stopping tolerance; number of sweeps over the grid on the fine grid and the coarse grid; relaxation method, etc. Trilinos's ML wraps the EPETRA matrix into an ML matrix with additional information about the aggregations at each level. Other interfaces are provided to set up the maximum levels, coarsening type, coarse grid solver, smoothers on each level, etc. Although some MGM solver parameters are common among AMG and GMG solver, for the AMG, most of time when the finest linear system is given the solver package can build the coarser systems internally. For GMG, the given domain information (grids and stencil) is used to construct the coarse grid, and the solver package can build the finer linear system internally. Based on discussion in Section 2.4, the required geometric information for GMG may need to be set with the matrix, so AMG and GMG interface can only share parameter setting methods while the matrix and right-hand side setup methods have to be different.

3 Design of LISI-MG

Because each MGM package has many application-dependent features and the interface and information they use differ greatly. LInear Solver Interface for Multigrid (LISI-MG) tries to abstract a minimal common set. Then subclassing can be used to support unique features or requirements they have. The interface serves as a common entry point for access to the different packages, and since the interface extends from earlier work [16], the application is able to access non-MGM solvers as well.

3.1 Solver Parameters

In the *lisi.SparseSolver* interface, generic solver parameter setup methods allow finer control on the solvers. Although multigrid solver parameters can be included to become part of this list of parameters, more specific parameters may have to be associated with the multigrid levels. At the minimum, an extra integer parameter is needed in the methods for levels in the hierarchy. The *MGMParameter* interface supports multigrid solver parameter setting. To clarify the boundary between multigrid and non-multigrid solvers, the *GeneralParameter* interface is extracted from the *lisi.SparseSolver* interface. As the name suggests *GeneralParameter* is for parameters associated with general solvers, such as *solvermethod*, *preconditioner*, *maxits* and *tol*. It supports generic setup routines and routines with more specific names, such as *setupTol*, *setMaxits*. *MGMParameter* sets up the maximum multigrid level, maximum number of pre/post-smoothing steps on each level, pre/post-smoothing methods on each level, stopping tolerance for each level of smoothing and the solver for the coarsest level of grid. The AMG and GMG interfaces might either encapsulate *MGMParameter* or inherit from *MGMParamter*.

Some parameter settings may have dependencies forcing a particular order, e.g., the *setMaxLevel* must be called before

setPreSmoother. Because the *setupMatrix* method generates the hierarchy grids, *setMaxLevel* must precede other solver related parameters. Defining *MGMParameter* requires separation into pre/post-parameter group, so LISI uses inheritance to avoid this and to also avoid extra work on the application side to handle object construction for *MGMParameter*.

3.2 Class Hierarchy

Section 2.5 shows that except for the parameters, the common set between AMG and GMG is small, so two separate interfaces are provided in the LISI-MG package. For medium-level interface, the *AMGSolver* expects the finest linear system input in three-array format. *AMGSolver* extends the *lisi.SparseSolver* interface (renamed as *lisi.SparseSolverWithMatrix*) with all its methods for setting up right-hand sides and the linear system. The GMG solver interfaces share the *MGMParameter* interface with AMG solver. *AMGSolver* and *GMGSolver* interfaces are on the same level of LISI's interface hierarchy. Because of the variance in task assignment that the application should provide, two solver interfaces do not share the matrix setup method. Figure 4 shows the interface hierarchy for LISI-MG. The *GeneralParameter* and *MGMParameter* interfaces are extended by the medium layer interfaces *AMGSolver* and *GMGSolver*, and the lower layer interface *lowMGMSolver* extends from *AMGSolver* to handle the operator-based MGM.

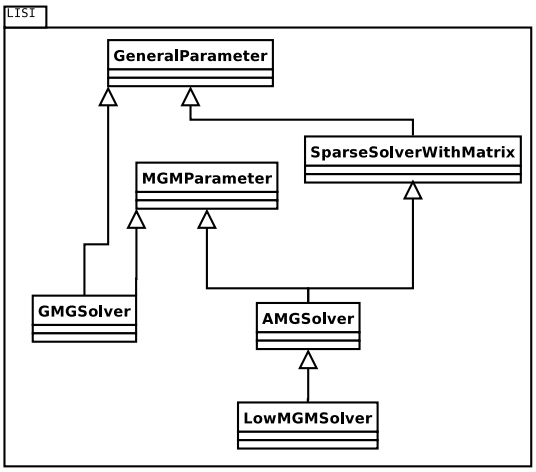

Figure 4. Class Hierarchy for LISI-MG Interface

3.3 AMG and GMG Interface

This interface is written in SIDL [14] to make clear that it is language independent, and the interface is shown in [15]. As discussed earlier, a *MGMParameter* interface includes all the parameter setting routines for general multigrid solvers. These parameters are common among HYPRE's BOOMERAMG and Trilinos's ML except for *setPreTol* and *setPostTol* methods. They have been added for completeness and is useful for the low-level interface. The *GMGSolver* and *AMGSolver* interfaces extend from the high level interfaces and are defined as a CCA **Port** which means they can be implemented by CCA **Component** and in turn can be exposed to other components as a CCA service.

A *GMGSolver* provides the GMG solver for the single structured grid. Since there is no uniform way to specify the complex geometry, some multigrid applications are developed bottom up without the clear boundary between solver and application as we discuss in 2.2. Our interface mainly focuses on the simple grid and expects the application to set up the linear system along with any required geometric information, such as grid and stencil. Our proposed interface supports only a single structured grid in the rectangle domain, which is the single common case supported by both PETSc and HYPRE. The grid information is set through *setGrid* method with the global index of mesh cell, The start and end nodes in the domain are specified as tuple. The stencil is set through *setStencil* with a array of length n in which $n = (\#neighbornodes + 1) \times dimension$. In general, a star-shaped stencil is used with 5 points for 2-D grids or 7 points for the 3-D ones. The *setupMatrix* and *setupRHS* assume that the grid and stencil are set up already. The application must fill the *values* array with the coefficients of the PDEs system. The methods *updateMatrixWithBoundary* and *updateRHSWithBoundary* are used to update the linear system and right-hand side with boundary conditions.

3.4 Low-Level MGM Interface

The low-level MGM interface provides an opeartor-based interaction between multigrid application and solvers per discussion in Section 2.2. Application sets up all the multigrid operators such as multilevel linear systems, prolongation and restriction operators. *lowMGMSolver* extends *AMGSolver* interface and works for both GMG and AMG since the operators are all algebraic defined. The overloaded method *setupMatrix* is to set up the multilevel coarser linear systems to solve. The extra parameter is used to indicate the *multigrid level*. Since the linear system is distributed (under block row partitioning assumption), four more methods need to be overloaded: *setStartRow*, *setLocalRows*, *setLocalNNZ* and *setGlobalCols*. These methods require the same extra parameter *multigrid level*.

Another method is added to handle the recursive calls for each V cycle, the method is exposed to the application, so that the application can control the outer loop by using its own convergence test. The *recurSolve* method assumes that the hierarchy grids are built during the initialization phase, so only the level number is needed to indicate the current level. Since ML and PCMG are using the different numbering to determine the level, this interface arbitrarily chooses that level 0 is the finest level, and level $n-1$ is the coarsest, where maximum number of levels is n.

There is an auxiliary interface *MGMOperator* is to be implemented by the multigrid application to provide *prolongation* and *restriction* operators. And it allows solver to callback to get the residual array transfered between fine and coarse grids.

4 Implementations

To validate this design, a simple reference implementation has been built for both AMG and GMG solvers using medium- and low-level interfaces. Because the emphasis is on the interface usability rather than on the scientifically-significant results, we take a test example as a two-dimensional Poisson equation:

$$u_{xx} + u_{yy} = f \tag{1}$$

in the unit square domain. For AMG interface tests, the boundary condition is defined as $x \times x \times sin(y)$ and $f = 0$, and the GMG interface test uses the 0 boundary conditions and $f = 1$. Both AMG and GMG tests use the stopping tolerance of 10^{-8} in the relative residual and the maximum allowed number of iterations is 200. All tests were run on the Linux cluster *odin* in the Computer Science Department at Indiana University. Odin has 128 nodes, each with two dual core AMD Opteron 2.0 Ghz processor and 4GB RAM on each computing node. Medium-level interface tests were performed using 16 processes while low-level interface tests were runing on 8 processes, and each result is the average of at least ten runs. The components are built with the CCA Tools [9] Version 0.6.0rc5 using a component generation tool Bocca [11] Version 0.5.0. All the packages and tools are compiled with GCC 3.4.6. HYPRE version is hypre-2.0.0, and Trilinos version is trilinos-8.0.2.

First we test the medium-level interface *AMGSolver* and *GMGSolver*.

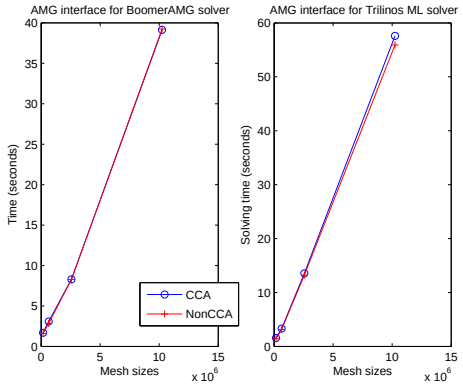

Figure 5. Test for AMG interface

The *AMGSolver* interface is implemented for HYPRE's BOOMERAMG solver through HYPRE's IJ system interfaces. A *HYPRE_IJMatrix* is created with finest linear system in COO format with global row/column indices. Then it is converted to *HYPRE_ParCSRMatrix* to make it ready for the solver. Similarly right-hand side and solution vectors are setup as *HYPRE_IJVector* with global row indices. The *Gauss-Seidel—Jacobi* hybrid relaxation is used for both post- and pre-smoother with three sweeps on each level. The maximum number of multigrid levels is 20. The tests are done for the mesh size of 400^2, 800^2, 1600^2, and 3200^2 (Figure 5 left). The blue line shows the results of calling HYPRE's BOOMERAMG solver directly, while the red line times the results of calling through *AMGSolver* interface. These two lines are almost overlapped. Thus, they demonstrate a negligible overhead when using the proposed interface.

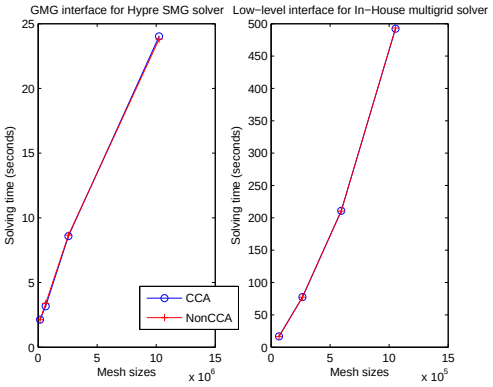

Figure 6. Test for GMG and Low-level interface

Trilinos ML solver is implemented with *AMGSolver* interface. Pre-smoother is *Jacobi* with three sweeps for all levels, and a *Gauss Seidel* smoother is used for post-smoother with three sweeps on each level. The coarsest solver uses *Gauss Seidel*. The maximum number of levels is 20 but *ML* modifies this value internally based on the matrix properties. The finest linear system is passed to the solver in COO format. The linear system is first setup as *Epetra_CrsMatrix*, and then converted to *MLMatrix*. A *ML_Aggregate* object is constructed within the implementation, and is hidden from application users. Problem sizes are the same as for the HYPRE BOOMERAMG test cases. The timing results (in Figure 5 right) shows a noticeable overhead only on the largest problem size of 3200^2.

To test the GMG interface, HYPRE's SMG structured grid based geometric multigrid solver was implemented with LISI-MG with maximum 100 iterations allowed and three pre- and post-smoothing steps. Figure 6 (left) shows a almost linearly increased running time versus the problem size (160000, 640000, 2560000, and 10240000). The overhead introduced is still negligible.

In order to test the low-level interface, an in-house MGM (*myMGM*) application is built for the test problem (1). The application constructs a hierarchical mesh discretization and the associated prolongation and restriction operators, then calls the low-level LISI-MG interface to get back the solution. The *lowMGMSolver* interface was implemented by setting up the multi-level linear system through *setupMatrix* method, and uses the methods from *MGMParameter* for optional multigrid solver parameters. Using the component model, the pre/post-smoothing solver can be easily changed for any solver package which complies with the LISI interface. Since the application side generates all the operators, it implements *MGMOperator* interface to provide callback functionality for restriction and prolongation. Solver side only needs to use the *MGMOperator* port. Figure 7 shows the general architecture for this multigrid solver component.

Tests for mesh sizes of 67081, 265225, 594441, and

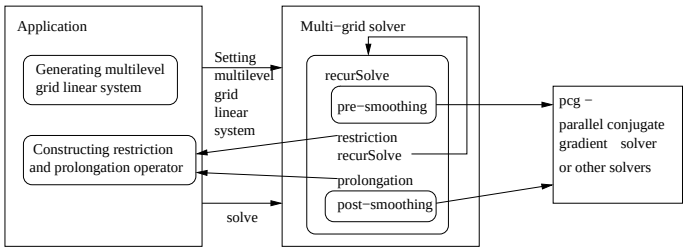

Figure 7. Test Architecture for Inhouse MGMSolver

1054729 mesh nodes show a running time that increases linearly with increasing mesh size. In Figure 6 (right), the *x*-axis is the size of meshes and the *y*-axis is the total running time for the whole solution process in seconds. On each level, the number of the relaxation steps is set as the square root of the linear system size to assure that each pre/post-smoothing solves the system to the specified stopping tolerance. Each run only needs two V cycles to reach the desired accuracy. The red and blue line show the timing results for *myMGM* built with and without CCA, respectively. For the small mesh size, the overhead is trivial.

Besides the lower overhead from using the proposed interface, the usage complexity is reduced greatly. In Table 1, four solvers are compared for their CCA and non-CCA invokation. The column `CCA` shows the number of code lines when calling the solvers via our proposed interface, the column `Non-CCA` gives the lines of code for direct solver invokation. The `Diff` column demonstrates the line reductions (the larger number means better complexity reduction).

Table 1. Usage Comparisons (in Code Lines) of Solver Packages

	CCA	Non-CCA	Diff %
In-House Multigrid Solver	37	216	82.8
Hypre BoomerAMG	32	90	64
Trilinos ML	32	80	60
Hypre SMG	133	243	45

5 Conclusion

In this paper, a suite of publicly available multigrid packages is investigated to abstract common interfaces. This work focuses on multigrid solvers due to their attractive performance for high-performance computing (HPC) simulations. Associating the component technology with multigrid solvers provides more choices for application users: Not only to choose a solver from the same package, but also to access solvers across different packages. A hierarchical solver interfaces corresponding to the levels of control of the application over the multigrid solver. The interfaces are designed for both geometric (GMG) and algebraic (AMG) multigrid solvers. The design of the former is more complicated, mainly because of non-uniform ways to specify the geometric information. We demonstrate that an interface can be designed for multigrid solvers. It enables an easy usage of current multigrid solver packages and leads towards a component based application-solver interaction framework.

References

[1] Hypre: Scalable Linear Solvers : high performance preconditioners. `http://www.llnl.gov/CASC/linear\_solvers/`, April 2008.

[2] M3D team. `http://w3.pppl.gov/m3d/index.php`, 2008.

[3] The ML API website. `http://trilinos.sandia.gov/packages/ml/mlapi.html`, April 2008.

[4] NIMROD. `https://nimrodteam.org/`, 2008.

[5] PETSc: Portable Extensibel Toolkit for Scientific C omputation. `http://www-unix.mcs.anl.gov/petsc/petsc-as/`, April 2008.

[6] Tops Solver Component. `http://www-unix.mcs.anl.gov/scidac-tops/solver-components/tops.html`, April 2008.

[7] The Trilinos Project. `http://software.sandia.gov/trilinos`, April 2008.

[8] W. L. Briggs, V. E. Henson, and S. F. McCormick. *A multigrid tutorial: second edition*. Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 2000.

[9] CCA-Forum. The DOE Common Component Architecture project. `http://www.cca-forum.org/`, April 2008.

[10] E. Chow, R. Falgout, J. Hu, R. Tuminaro, and U. Yang. A survey of parallelization techniques for multigrid solvers. *Parallel Processing for Scientific Computing, M.A. Heroux, P. Raghavan, and H.D. Simon, eds., SIAM Series on Software, Environments, and Tools*, 2006.

[11] W. R. Elwasif, Boyana R. Norris, Benjamin A. Allan, and Robert C. Armstrong. Bocca: a development environment for HPC components. In *Proceedings of the 2007 symposium on Component and framework technology in high-performance and scientific computing*, pages 21–30, Montreal,Quebec,Canada, October 21 -22 2007.

[12] R. D. Falgout. An introduction to algebraic multigrid. *Computing in Science and Engineering*, 8:24–33, Nov/Dec 2006.

[13] M. Gee, C. Siefert, J. Hu, R. Tuminaro, and M. Sala. ML 5.0 smoothed aggregation user's guide. Technical Report SAND2006-2649, Sandia National Laboratories, 2006.

[14] S. Kohn, G. Kumfert, J. Painter, and C. Ribbens. Divorcing Language Dependencies from a Scientific Software Library. In *10th SIAM Conference on Parallel Processing*, Portsmouth, VA, March 12-14 2001. LLNL document UCRL-JC-140349. See also `http://www.llnl.gov/CASC/components/babel.html`.

[15] F. Liu. LISI-MG: CCA interface for multigrid solvers. `http://www.cs.indiana.edu/~fangliu/lisi-mg.sidl`, October 2008.

[16] F. Liu and R. Bramley. CCA-LISI: On Designing a CCA Parallel Sparse Linear Solver Interface. In *Proc. 21th Int'l Parallel & Distributed Processing Symp.(IPDPS)*. ACM/IEEE Computer Society, Long Beach, CA, 2007. 10 pages.

[17] M. Sosonkina, F. Liu, and R. Bramley. Usability Levels for Sparse Linear Algebra Components. *Concurrency and Computation: Practice and Experience*, 20:1439–1454, 2008.

[18] R. S. Tuminaro and C. Tong. Parallel smoothed aggregation multigrid : Aggregation strategies on massively parallel machines. In *Proceedings of ACM/IEEE SC 2000 Conference*, pages 5– 5, November 2000.